Automated Task-Based Synthesis and Optimization of Field Robots

Chris Leger

The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
email: blah@cmu.edu
phone: (412) 268-8157
fax: (412) 268-5895

John Bares

National Robotics Engineering
Consortium
Carnegie Mellon University
#10 40th Street
Pittsburgh, PA 15201
email: bares@cs.cmu.edu
phone: (412) 268-7091
fax: (412) 681-6961

Abstract

We present Darwin2K, a widely-applicable, extensible software tool for synthesizing and optimizing robot configurations. The system uses an evolutionary optimization algorithm that is independent of task, metrics, and type of robot, enabling the system to address a wide range of synthesis problems. A representation for robot configurations is described which enables manipulators and mobile robots (including free-flying robots, mobile manipulators, modular robots, and multiple or bifurcated manipulators) to be synthesized. Darwin2K includes a toolkit of simulation and analysis algorithms which are useful for many synthesis tasks; some of these components, such as dynamic simulation, are novel in automated synthesis of robots. An extensible system architecture enables new synthesis tasks to be addressed while maximizing use of existing system capabilities; this extensibility is a key contribution of the system. We apply Darwin2K to a robot synthesis tasks that includes synthesis and optimization of robot kinematics, dynamics, structural geometry, and actuator selection.

Introduction: Why Automated Synthesis?

Robot configuration design is often performed in an ad hoc manner. It can be difficult to translate the requirements of a novel task into a robot configuration, and in many robot design problems there are no general design heuristics to provide guidance. Human designers rely on intuition and experience with related design problems, and on engineering rules based on the experience of other expert designers. While these methods are suitable for human engineers and can lead to designs that are well-understood and can be predicted to meet the design objectives, one can argue that human design approaches greatly restrict the range of possible designs that are explored. When addressing an entirely new problem, a designer may not have much of a relevant experience base to draw upon; in this case, an automated configuration synthesis tool can generate well-optimized solutions without requiring previous experience, and can explore much more of the design space than a human designer.

Typically, the configuration process generates the overall form of the robot, including kinematics and other geometry at the bare minimum but often including approximate descriptions of inertial properties, actuator and material selection, and structural geometry. Frequently, a human investigates a small number of concepts on paper and selects a few that look promising. More detailed studies may then be performed on these, culminating in the simulation of one or more designs. One of the candidates is selected for detailed design, with tools such as finite element analysis used to evaluate parts of the design. Once the robot is built, changes may be required due to unforeseen problems or factors that were not modeled in simulation. Significant design iterations are often not practical, since much of a project's schedule and resources may be devoted to creating a single robot; building a second or third robot to remedy design flaws is out of the question for many large robot design projects. Thus, it is crucial to perform as much analysis and simulation before the robot is built, and it is highly desirable to "get it right" the first time -- since the first time may be the only time.

Because of these factors, automated synthesis tools are especially attractive for robot design. Synthesis tools can address novel design problems, explore a large number of designs, quickly perform design iterations in simulation, and produce a well-optimized solution with high confidence of performance, all of which contribute to the likelihood of success of the first physical implementation.

Darwin2K is a software toolkit for automated synthesis. It includes capabilities for quickly describing and modifying robot configurations, simulating configurations as they perform tasks, and automatically synthesizing configurations to meet task-specific requirements and to optimize performance. Darwin2K is very useful in the early stages of the configuration process, as it can automatically explore tens of thousands of designs and can allow a human designer to rapidly perform design iterations and evaluate potential robots in simulation.

Related Work

There has been much prior work in the area of robot design. However, research in *automated* design--that is, the development of software systems which perform a significant part of the synthesis of a robot--has been limited to a handful of systems with widely

varying scope and goals.

Most configuration synthesis systems have employed evolutionary algorithms of some sort, due to the robustness of such algorithms to local minimal and to search spaces that are highly nonlinear and of varying dimension. Previous approaches to robot configuration synthesis can be divided into two categories: modular design and non-modular design. The former group is characterized by constructing robots from fixed modules, mirroring a set of reconfigurable hardware modules from which the actual robot is built. The latter group has synthesized robots that are not built from fixed modules, but which consist of purely kinematic descriptions (i.e. Denavit-Hartenburg parameters or equivalent).

Much of the previous work in synthesizing robot configurations has been in the area of modular robots. Most of these have been for modular manipulators ([Ambrose94], [Chen95], [Paredis96], [Chocron97], [Han97]), while one ([Farritor96]) addressed mobile robots. Several of these approaches stand out for their unique contributions: [Ambrose94], while limited to planar manipulators, included modeling of actuators and link deflection. [Paredis96] used higher-fidelity simulation (including collision detection for non-trivial geometry) and generated fault-tolerant manipulators and motion plans. [Farritor96] generated modular field robot configurations and motion plans, though a low-fidelity evaluation process was used in the interest of decreasing computation time.

Other work has addressed kinematic configuration of monolithic (non-modular) manipulators. Kim and Khosla [Kim93] use a genetic algorithm to determine the location and Denavit-Hartenberg parameters for a manipulator. Chedmail and Ramstein [Chedmail96] use a genetic algorithm to determine the type (one of several manipulators) and location of a robot to optimize workspace reachability. McCrea [McCrea97] discusses the application of genetic algorithms to the selection of several parameters for a manipulator used in bridge restoration.

Several key limitations are apparent in previous robot synthesis systems. The only systems to consider non-kinematic properties were those using fixed modules; the systems for monolithic manipulators are purely kinematic and have limited robot representations--specifically, monolithic manipulators have been modeled as joints connected by zero-thickness line segments. The strictly modular systems, while able to represent non-kinematic properties, are not able to independently vary properties such as actuators or link structure, thus leading to suboptimal solutions. Previous systems have not addressed analysis and simulation needs such as dynamic simulation and estimation of link deformation due to loads, thus inhibiting the ability of these systems to meaningfully optimize actuator selection or link structural geometry and inertial properties.

System Description

An earlier version of Darwin2K is described in [Leger98]. Here, we will briefly discuss Darwin 2K's architecture and describe recent additions, including an improved method for optimizing multiple metrics, dynamic simulation, and optimization of non-kinematic properites.

Darwin2K consists of two distinct parts: a configuration synthesizer, and a simulation and analysis tool. The synthesizer, called the Evolutionary Synthesis Engine (ESE), uses the simulation tool to evaluate the performance of each new configuration it creates. The ESE's evoluationary algorithm synthesizes robot configurations by applying genetic operators to one or more existing parent configurations. The ESE selects the parents from a population of configurations according to their performance, so that each new configuration is likely to perform well and will occasionally outperform its parent configuration(s). The performance of each robot is measured in a task-specific manner: the robot performs the task in simulation, and multiple task-specific metrics (selected by the designer) record various aspects for the robot's performance. Darwin2K's 15 existing metrics include power consumption, task completion time, robot mass, stability, actuator saturation, and collision measurement.

Each configuration is assembled from one or more *parameterized modules*, of which there are four basic types: bases, links, joints, and tools. Each module represents a part of a robot, with an arbitrary number of parameters that dictate specific properties. Modules can have varying complexity, ranging from a simple link with no moving parts, to a joint with one or degrees of freedom, to an entire manipulator, to a mobile base. Each module can have a number of connectors, which indicate how the module can be attached to other modules. A module's parameters may describe any property of the module, such as a geometric dimension, a discrete component selection (e.g. a motor, gearhead, or material choice), or a controller gain. All modules have the same generic interface to the ESE so that new module types can be added to the system without requiring changes to the ESE, thus enabling Darwin2K to use task-specific modules when addressing novel design problems. Darwin2K currently contains approximately 30 general-purpose parameterized modules, as well as 10 or so task-specific modules that were created for specific synthesis problems (see Figure 1 for examples).

To represent a particular robot configuration, one or more parameterized modules are assembled into a graph, called the Parameterize Module Configuration Graph (PMCG). Each node in the graph is a parameterized module, and each edge is a connection between modules. For serial-chain manipulators, this graph is simply a connected series of modules, usually starting with a base, followed by one or more links or joints, and ending with a tool. Multiple and branching manipula-

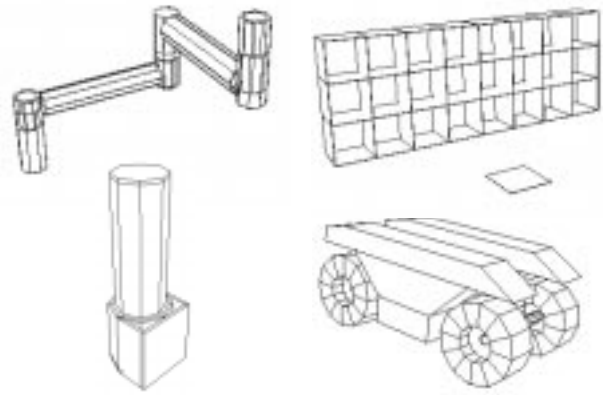

Figure 1: A selection of Darwin2K modules. Clockwise from upper left:
• a SCARA module with parameters for motors, gearboxes, material, link lengths, link diameters, wall thicknesses, and joint angle offsets
• a base module with bins for stacking payloads; parameters for aspect ratio, number of bins, and access location
• a mobile base for construction robots, with parameters for wheelbase, tread, and attachment location.
• a right-angle joint module with parameters for motor, gearbox, material, link length, diameter, and wall thickness

tors can also be represented by having multiple branches in the graph. One important aspect of the PMCG is that each connection , and each module parameter, have a *const-flag*, which can be set to indicate to the ESE that a particular feature should not change. Thus, if the designer knows of any task-specific features that would be beneficial, they can be directly encoded and preserved by the synthesizer..

The PMCG representation is easy for a human designer to manually modify, thus allowing the designer to quickly build configurations that can be simulated using Darwin2K's simulation tool. In addition to displaying simulations of robots performing tasks, the simulator also operates in a no-display mode, so that the synthesizer can use it to measure the performance of configurations. Many simulators can be distributed over a network of computers, all providing fitness measurements to the synthesizer. The simulator has a wide range of capabilities including kinematic and dynamic simulation, collision detection (using the RAPID library [Gottschalk96]), motion planning and estimation of link deflections. Darwin2K's simulation toolkit includes several controllers, a motion planner for planar mobile robots, and several trajectory representations, and also allows the addition of new, task-specific components such as controllers, performance metrics, modules, and analysis algorithms. With these existing components and the ability to add new capabilities, a designer can quickly create a relevant simulator for a new application and can begin manual or automated synthesis of an appropriate robot.

Task Specification

Darwin2K performs task-specific synthesis: robots are synthesized for a specific task, and are evaluated with respect to the performance metrics and requirements of the task. There are two components to the task specification: a set of performance requirements or metrics, and a description of how the task is performed. The vast majority of design problems involve multiple, often-conflicting metrics such as cost versus performance, speed versus power, or mass versus capability. Additionally, most tasks have hard limits that must be met for one or more metrics: there can be no collisions during operation, mass must be under a certain limit, etc. These requirements and non-linear dependencies (i.e. power consumption is irrelevant if a robot can't complete the task) cannot be effectively encoded by simple scalarization, such as using a weighted sum of metrics to create a single measure of performance. To remedy this, Darwin2K's task specification includes these constraints and priorities in a natural manner, called Requirement Prioritization. The designer selects a number of metrics (typically between 3 and 10), ranks them in order of importance, and sets an acceptability threshold for some. For example, a free-flying space robot tasked with servicing a satellite might have the following metric specification:

- 100 % task completion
- 0 collisions
- < 3mm cross-track error at the tool endpoint
- average actuator torque < 80% of rating for each actuator
- link deflection < 1mm
- minimal mass, energy, and task completion time.

Task completion, collision, and accuracy are most important: any configuration that does not satisfy these is inferior to any configuration that does. Thus, these are given a priority of 0 (most important). To ensure that actuators and links are appropriately sized, the actuator torque and link deflections are added next, with a priority of 1. Finally, we would like to minimize mass, energy consumption, and completion time given that the other requirements are satisfied, so these are given a priority of 2. Darwin2K uses this specification to determine how configurations are selected for reproduction: the highest-priority metrics (completion, collision, and accuracy) are used until the population contains a certain number of configurations that meet the acceptability thresholds; then the next-highest group is used until enough configurations meet that group's thresholds; and finally the last group of metrics (mass, energy and time) are used until the synthesis process ends (since no acceptance criteria were set for them). The synthesis process halts when either a time limit is reached or when a fixed number of configurations have been generated (typically between 40,000 and 160,000). While optimizing within each group of metrics, the metrics are randomly selected for

use based on how many configurations in the population satisfy each metric. Thus, the metrics whose acceptability thresholds are satisfied by few or no configurations will be used to select configurations more frequently than the metrics which many configurations satisfy. This automatically guides the synthesizer to remedy the weaknesses of the population as a whole. In practice, we have found this method to be much more effective in optimizing multiple metrics than scalarization approaches.

Specifying requirements and metrics is only one part of the task specification; it is also necessary to describe the actions to be performed by the robot. For robots with manipulators, this typically involves describing trajectories and any payloads or endpoint forces for each manipulator. Parameters for each simulation component are also set at this point.

Darwin2K contains a generic simulator for one or more manipulators following a series of endpoint trajectories using a Jacobian controller based on the Singularity-Robust Inverse (SRI) [Nakamura86]. While this simulator is suitable for a range of typical manipulation tasks, each target application may have unique simulation and evaluation requirements. For this reason, we have structured Darwin2K's software architecture so that a new, task-specific simulator can be quickly constructed using Darwin2K's existing simulation components, which include collision detection, several controllers, payload models, a motion planner for planar mobile robots, and several trajectory representations.

Task-specific simulators usually require little coding, normally just a main simulation loop: the evaluation components handle the numerical details, while the main simulator just controls which simulation capabilities and controllers are used over the course of the simulation. This allows the designer to rapidly construct a simulator tailored to the application at hand, ensuring that Darwin2K can accurately and relevantly measure a robot's performance. While Darwin 2K's simulation capabilities are useful by themselves to the designer, once the simulation has been created Darwin2K can perform synthesis and optimization of all or part of a robot essentially for free--the designer does not have to make any modifications to the synthesis engine.

Dynamic Simulation

While a purely kinematic simulation may be adequate for tasks that do not require large forces to be applied by or to the robot, there are some tasks for which dynamic simulation must be used to accurately assess a robot's performance. Two broad classes of robots for which dynamic simulation is crucial are free-flying robots (either in space or underwater), and large, high-force robots such as those targeted at construction applications. Free-flying robots experience reaction forces during manipulation that may move the entire robot, while construction robots frequently require one or more actuators to be operated at saturation (i.e. providing maximum force or torque output), in which case the response of the robot will not be known without a dynamic model.

Darwin2K supports forward and inverse dynamic modelling for fixed-base, mobile, and free-flying manipulators, including robots with branching or multiple manipulators. The inverse dynamic problem (computing joint torques required to produce desired joint accelerations) is relatively easy to solve using the well-known iterative Newton-Euler dynamic formulation [Craig89]. However, the forward dynamic problem--computing the motion of a robot given the force applied at each degree of freedom--is more difficult. The dynamic model of a robot can be expressed by the equation

$$T = M(\Theta)\ddot{\Theta} + V(\Theta, \dot{\Theta})$$

Where T is a vector of joint torques, $M(\Theta)$ is the *mass matrix*, $V(\Theta, \dot{\Theta})$ is a vector of inertial and applied forces, and Θ is a vector of joint positions. Forward dynamic simulation requires that M and V be constructed from the current configuration of the robot, after which $\ddot{\Theta}$ is found by solving the set of equations. Computing M and V is the crux of the problem: the Newton-Euler equations (or equivalent) must be applied symbolicly to yield equations for each element of T, and then these equations must be factored to pull out the coefficients for each entry of $\ddot{\Theta}$. Terms in the equations that do not contain a joint acceleration get lumped into V. While this can be easily done using Mathematica or a similar package, or even by hand for a known robot, these approaches are not very useful for automated synthesis where efficient dynamic simulation must be performed for the tens to hundreds of thousands of unique robots created by the synthesizer during a synthesis run.

Our approach to this problem is based on treating each scalar in the dynamic equations as a linear combination of joint accelerations and a constant term; this is possible because the dynamic equations do not contain products of multiple Θ_is. To fill the ith row of M and V, we symbolically compute the equation for the ith joint torque, evalute the equation numerically, and then extract the coefficient of each joint acceleration and the constant coefficient. Performing this process for each torque equation (i.e. for each row of M and V) gives a system of equations that can be solved for $\ddot{\Theta}$ given the torque or force applied at each joint (T).

We have implemented a set of C++ arithmetic and vector classes specialized for this approach. As described above, each scalar value is considered to be a linear combination of joint accelerations and a constant term; we call these *s-vals*, for Separated Values (since each joint acceleration coefficient is accumulated separately). The Newton-Euler equations are symbolically evaluated once using these classes to yield an equation for each joint torque. At each simulation time step, these equations are then numerically evaluated based

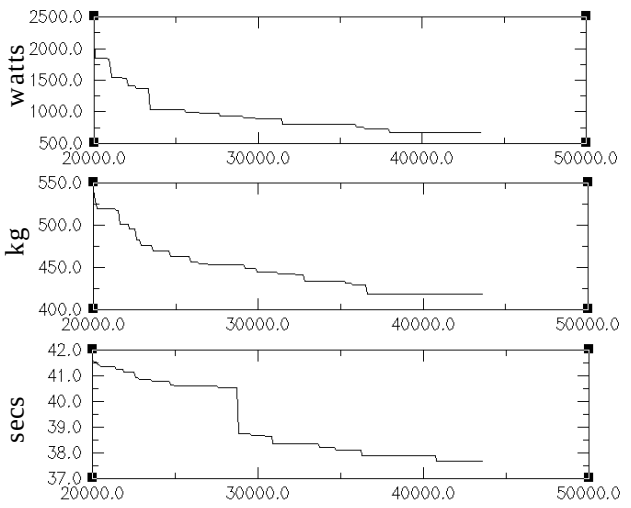

Figure 2: (top to bottom) Best mass, energy, and time of feasbile configurations during optimization

on the current position and velocity of each joint. Each computed joint torque (i.e. element of *T*) is an s-val; the s-val's elements are the elements of the corresponding rows in *M* and *V*. After numerically evaluating the s-val for each joint torque equation (i.e. for each row of *T*, *M*, and *V*), we solve the system of equations for $\ddot{\Theta}$. We then numerically integrate $\dot{\Theta}$ and $\ddot{\Theta}$ using the Runge-Kutta 4 algorithm [Press92] to compute the next state of the robot. We use an adapative stepsizing algorithm to maximize numeric stability and computational efficiency.

Design Examples

To verify Darwin2K's ability to perform configuration synthesis for challenging applications, we used Darwin2K to configure a free-flying space robot with two manipulators for a satellite servicing task. The task is loosely based on the requirements of the Ranger Telerobotic Flight Experiment and Telerobotic Shuttle Experiment ([SSL99]), and consists of both kinematic and dynamic simulations. The kinematic simulation is designed to ensure that the robot's manipulators have adequate workspaces, and consists of having the manipulators follow trajectories that cover a reasonable workspace. The SRI controller is used to control the manipulators, and collision detection is performed to ensure that there is no interference between the robot's manipulators.

The dynamic simulation is more involved and is designed to be representative of the operations required of the robot. The robot starts at a position near a 9100kg satellite, uses one arm to grapple the satellite and then move the base to a known location relative to the satellite. The robot's other manipulator (the work manipulator) then removes an Orbit Replacable Unit from the satellite, and the grapple manipulator moves the base relative to the satellite again. Finally, the work manipulator re-inserts the ORU into the satellite at another location. Darwin2K's free-flyer controller (an SRI controller augmented with the robot's dynamic model) was used to control the robot during the dynamic simulation. The ORU has a mass of 5kg and requires a torque of 14 Nm (10 ft-lbs) to engage or disenage from the satellite, and 45N (10lbs) of force to insert or remove. The metrics used were those specified earlier in the "Task Specification" section. During optimization of the first requirement group (task completion, collision detection, and accuracy), kinematic simulation was used for the entire simulation; dynamic simulation was used as specified above for the remaining two requirement groups.

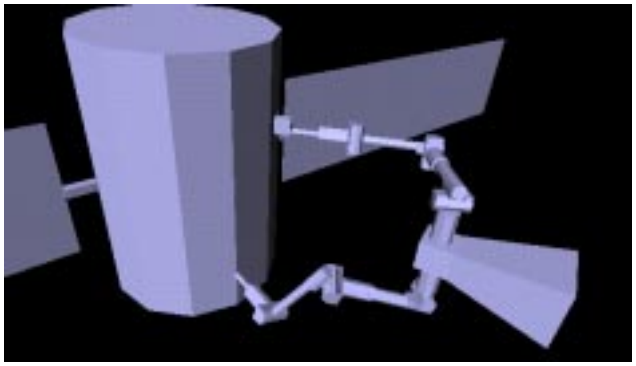

Figure 3: Screenshot from Darwin2K's simulator showing the feasible configuration with lowest energy consumption as it begins to insert the ORU into the satellite.

Five module types were used in the synthesis process: a free-flying base module massing 267kg; a tool module representing Ranger's Microconical End Effector (MEE); and three joint modules (each with six parameters for various dimensions and for motor, gearbox, and material selections). The robot was constrained to have symmetric arms with 7 or 8 degrees of freedom each, and in addition to the parameters of each module, there were six task parameters: two specified the base's initial pose relative to the satellite, and the remaining four specified the velocity and acceleration of the base during each of the two relative base motions. The design space contained 5.14×10^{58} configurations. For this problem, we ran Darwin2K using idle CPU time from approximately 30 computers over 8 hours. After evaluating 5,000 configurations, 100 out of 200 configurations in the population satisfied the first requirement group. After 11,000 evaluations, a configuration satisfying the second requirement group had been created, and Darwin2K moved to the final requirement group at 20,000 configurations. At this point, 16 of the 200 configurations in the population satisfied the task requirements; the remainder of the run was spent minimizing mass, energy, and task completion time. Figure 2 shows the best mass, energy, and time of the feasible configurations as the final requirement group was optimized. During the last requirement group, the mass of the lightest configuration in the population decreased from 541kg to 418kg; the lowest energy consumption decreased

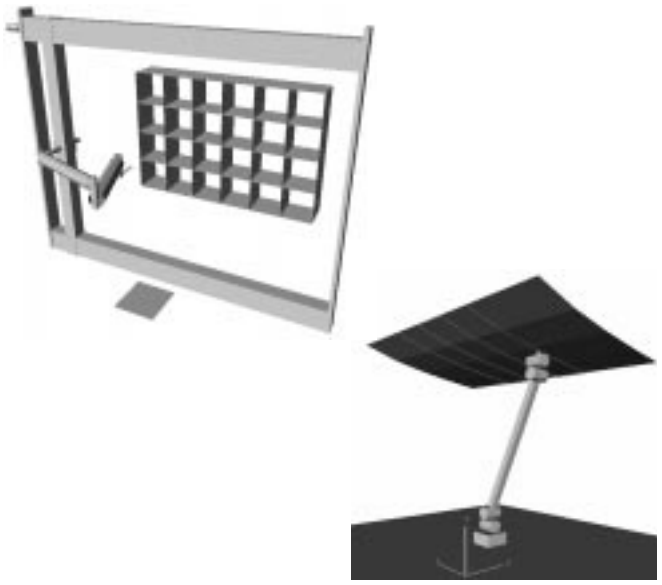

Figure 4: A gantry-mounted SCARA manipulator synthesized for a storage and retrieval task (upper left), and a 5-DOF arm with prismatic joint synthesized for waterproofing tiles on the Space Shuttle (lower right).

from 2.1kJ to 0.6kJ, and task completion time decreased from 41s to 37s. Figure 3 shows the feasible configuration with lowest energy consumption. After 8 hours, 43,000 configurations had been evaluated; on average, each evaluation took 20 seconds of realtime for roughly 40 seconds of simulated time.

We have also applied Darwin2K to other synthesis tasks where dynamic simulation is not required. In these cases, actuator selection and link structural sizing can still be performed since the inverse dynamic equations can be applied at each time step to estimate the torque required at each actuator. These torques can be compared to the torque capability of each actuator, and can be used to estimate link deflection to ensure that links are sufficiently stiff. Figure 4 shows results from two of these other synthesis tasks: a robot for storing and retrieving containers of wafers in semiconductor fabrication plant and a manipulator for waterproofing Space Shuttle tiles (based on the task described in [Kim93]. While the details of these synthesis tasks are beyond the scope of this paper, they are mentioned here to give the reader an idea of the breadth of application of Darwin2K.

Conclusion

Darwin2K is a practical, widely-applicable task-based automated synthesis system, and is significantly more capable than previous approaches to automated synthesis. The system's extensible, modular software architecture allows task-specific simulations to be easily constructed so that new tasks may be addressed in a relevant manner. The Parameterized Module Configuration Graph can represent a wide range of robot configurations and allows the designer to quickly generate prototypes and evaluate them in simulation. Darwin2K's synthesis engine can optimize partially-specified configurations and synthesize entirely novel robots, allowing well-optimized designs to be generated for novel, complex, and poorly-understood applications.

References

[Ambrose94] R. Ambrose and D. Tesar. The Optimal Selection of Robot Modules for Space Manipulators. In *Proceedings of the ASCE Space 94 Conference*.

[Chedmail96] P. Chedmail and E. Ramstein. Robot mechanism synthesis and genetic algorithms. *In Proceedings of ICRA 1996*, pages 3466–3471.

[Chen95] I. Chen and J. Burdick. Determining task optimal modular robot assembly configurations. *In Proceedings of ICRA 1995*, pages 132–137, 1995.

[Chocron98] O. Chocron and P. Bidaud. Genetic Design of 3D Modular Manipulators. In *Proceedings of ICRA 1997*, pp. 223-228.

[Craig89] J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison-Wesley, 2nd edition, 1989.

[Gottschalk96] S. Gottschalk, M. C. Lin and D. Manoch. *OBB-Tree: A hierarchical Structure for Rapid Interference Detection.* Technical Report TR96-013, Department of Computer Science, University of North Carolina, Chapel Hill, 1996.

[Han98] J. Han, W. K. Chung, Y. Youm, and S. H. Kim. Task Based Design of Modular Robot Manipulator using Efficient Genetic Algorithm. *In Proceedings of ICRA 1997*, pp. 507-512.

[Kim93] J.-O. Kim and P. Khosla. Design of space shuttle tile servicing robot: An application of task-based kinematic design. In *Proceedings of ICRA 1993*, pp 867–874.

[Leger98] C. Leger and J. Bares. Automated Synthesis and Optimization of Robot Configurations. In *Proceedings of the 1998 ASME Design Engineering Technical Conferences*.

[McCrea97] A. McCrea. Genetic algorithm performance in parametric selection of bridge restoration robot. In *Proceedings of the 14th International Symposium on Automation and Robotics in Construction*, pages 437–441, 1997.

[Nakamura86] Y. Nakamura. Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of Dynamic Systems, Measurement, and Control*, 108:163–171 September 1986.

[Paredis96] C. Paredis. *An Agent-Based Approach to the Design of Rapidly Deployable Fault Tolerant Manipulators*. PhD thesis, The Robotics Institute, Carnegie Mellon University, 1996.

[Press92] W. Press and S. Teukolsky and W. Vetterling and B. Flannery, *Numerical Recipes In C*. Cambridge University Press, 2nd edition, 1992.

[SSL99] University of Maryland Space Science Laboratory's Ranger project homepage: http://www.ssl.umd.edu/homepage/Projects/ranger.html (as of July 1999).